Using linear logic and proof theory to unify computational logic

Dale Miller

▶ **To cite this version:**

Dale Miller. Using linear logic and proof theory to unify computational logic. Proceedings of Trends in Linear Logic and Applications (TLLA 17), Sep 2017, Oxford, United Kingdom. hal-01615673

HAL Id: hal-01615673
https://hal.science/hal-01615673

Submitted on 13 Oct 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Using linear logic and proof theory to unify computational logic

by Dale Miller, Inria

To be presented at
TLLA 17: Trends in Linear Logic and Applications
3 September 2017, Oxford, UK

The opening sentence of Girard's 1987 paper introducing linear logic contains the words "Linear logic is a logic behind logic". In this talk, I will explore a slight variation of this phrase: "Linear logic is the logic behind computational logic". Switching the indefinite article into a definite article is, of course, dangerous since we do not know what future analysis might reveal. I retain this switch here since, for the scope of this talk, I will not be tempted to consider any alternative to the core principles of linear logic. Also, this latter phrase narrows "logic" to just "computational logic". By this term, I mean, the discipline that has evolved within theoretical computer science for which there are now several conferences (such as LICS and CSL) as well as journals (such as the *Transactions on Computational Logic*). In particular, I assume that computational logic includes the following topics:

- Logic programming (proof search)
- Term representation
- Type systems (Curry-Howard)
- Functional programming (proof normalization)
- Model checking
- Theorem proving in logic and arithmetic, both automatic and interactive.

Every one of these topics is mature (having been studied now for decades) and broadly applicable (having numerous applications in practice or within computational logic itself). Every one of these topics is taught routinely to undergraduate or graduate students interested in applying formal methods to computer science problems.

Young people approaching topics in computational logic will find texts and articles that develop these topics in isolation from each other and with antipathy for one another. For example, intuitionistic logic was developed to rid (classical) logic of certain logical principles and model checking was

born in the 1980's as an anti-proof movement (against Floyd-Hoare style proofs).

In this talk, I want to address the challenge of finding a core set of principles that could provide a foundation for all of these topics and, as such, allow students to understand principles that are common and to appreciate the different reasons those general principles need to be differentiated into these many, separate topics.

Our proposed common foundation includes the following three logics.

1. MALL, with its classification of connectives as being additive or multiplicative and with having unambiguous polarities.
2. Linear logic (LL), as an extension of MALL with exponentials (and possibly with first-order and higher-order quantification). LL can serve as a foundation for classical and intuitionistic logics.
3. μMALL, as an extension of MALL with fixed points instead of exponentials. This logic appears to be more appropriate than LL as a foundation for model checking and inductive (and coinductive) theorem proving.

The foundation of this development is built on the theory and structure of proofs (i.e., on proof theory). Here, sequent calculus is central. While many people feel that sequent calculus is just one choice of proof presentation (among other choices such as natural deduction, tableaux, Herbrand instances, dependently typed λ-terms, etc), I will promote the point that sequent calculus should be seen as having a distinguished position: it provides the "assembly language" for realizing proof systems. The amorphous nature of Gentzen's sequent calculus can be tamed and exploited by the use of focused proof systems. The notion of synthetic inference rules arising from focused proofs provides a framework that helps to link and distinguish the many topics within computational logic.

A corollary of our approach to organize computational logic is that topics such as model theory, category theory, type theory, algebra, and game semantics do not play a role in presenting a unified approach to computational logic. Instead, those topics can be expected to provide deep results and historical connections in various specialized domains of computational logic.

Draft: 21 July 2017